Examen de Prácticas de Programación

1. Cuestiones
   1. (0.5 puntos) Se dispone de la clase Punto2D que compila correctamente:

**public class** Punto2D { **public** int x; **public** int y;

**public** Punto2D(int x, int y) {

**this**.x = x;

**this**.y = y;

}

}

Se pretende compilar la clase Punto3D que extiende a Punto2D:

**public class** Punto3D **extends** Punto2D {

**public** int z;

}

La compilación de la clase Punto3D produce el error:

"cannot find symbol - constructor Punto2D()"

("No se encuentra el símbolo - constructor Punto2D()").

Explica la causa del error y propón la modificación de la clase Punto3D que lo corrige.

***Falta añadirle el constructor, cuando una clase hereda de otra, subclase, tiene la obligación de crear el método constructor, con los atributos del padre + lo suyos propios).***

***public class Punto3D extends Punto2D {***

***public int z;***

***}***

***Public Punto3D(int x, int y, int z){***

***Super(x,y);***

***this.z = z;***

***}***

* 1. (0.5 puntos) Describe el error de compilación que se produciría en la línea marcada con la flecha:

**try** {

... *// instrucciones no relevantes para la pregunta planteada*

} **catch** (Exception e) {

... *// instrucciones no relevantes para la pregunta planteada*

} **catch** (ArithmeticException e) { 

... *// instrucciones no relevantes para la pregunta planteada*

}

***ArithmeticException: Capturas posibles errores de ejecución, en este caso, cuando hace el try, si da un error de este tipo la captura. Algun ejemplo es dividir por cero, es cuando no puede resolver una operación matemática.***

* 1. (1 punto) En el código mostrado a continuación:
     + Indica las cláusulas "throws" necesarias para que compile correctamente.
     + Indica la salida por consola que se produciría si se ejecuta el programa con "CASO=O", "CASO=1" y "CASO=2".

public class LanzaExcepciones {

private static final int CASO=...;

(puede valer O, 1 o 2)

public static class MiExcepción extends Exception {}

public static class MiRuntimeExcepción

extends RuntimeException{}

private static void método3(int caso) { switch (caso) {

case O: throw new MiExcepción();

case 1: throw new MiRuntimeExcepción();

}

}

private static int método2() { int ret = 1;

try {

System.out.println("2:antes"); método3(CASO); System.out.println("2:después"); return 2;

} catch (RuntimeException e) {

System.out.println("2:catch");

} finally {

System.out.println("2:finally");

}

System.out.println("2:final"); return ret;

}

private static int método1() { int ret = O;

try {

System.out.println("1:antes"); ret = método2(); System.out.println("1:después");

} catch (MiExcepción e) {

System.out.println("1:catch");

}

System.out.println("1:final"); return ret;

}

public static void main(String[] args) {

System.out.println("main:antes"); int ret = método1();

System.out.println("main:después. ret:"+ret);

}

}

Falta añadir la clausula ***throws***  a los métodos para que pueda capturar las excepciones, dependiendo el ***CASO.***

public class LanzaExcepciones {

private static final int ***CASO***;

public static class MiExcepción extends Exception {}

public static class MiRuntimeExcepción extends RuntimeException {}

private static void método3(int caso) throws MiExcepción, MiRuntimeExcepción {

switch (caso) {

case 0:

throw new MiExcepción();

case 1:

throw new MiRuntimeExcepción();

}

}

private static int método2() throws MiExcepción, MiRuntimeExcepción {

int ret = 1;

try {

System.***out***.println("2:antes");

*método3*(***CASO***);

System.***out***.println("2:después");

return 2;

} catch (RuntimeException e) {

System.***out***.println("2:catch");

} finally {

System.***out***.println("2:finally");

}

System.***out***.println("2:final");

return ret;

}

private static int método1() throws MiExcepción, MiRuntimeExcepción {

int ret = 0;

try {

System.***out***.println("1:antes");

ret = *método2*();

System.***out***.println("1:después");

} catch (MiExcepción e) {

System.***out***.println("1:catch");

}

System.***out***.println("1:final");

return ret;

}

public static void main(String[] args) {

System.***out***.println("main:antes");

int ret = 0;

try {

ret = *método1*();

} catch (MiExcepción | MiRuntimeExcepción e) {

// Manejo de excepciones en el método main

}

System.***out***.println("main:después. ret:" + ret);

}

}

1. (4.25 puntos) Completa el diseño e implementación de la aplicación (únicamente de las partes solicitadas) que verifica los siguientes requisitos:

Se desea gestionar el préstamo de los documentos de una biblioteca. Existen dos clases de usuarios de la biblioteca: los socios y los usuarios ocasionales. Los socios pueden tener prestados simultáneamente 20 documentos como máximo, mientras que los clientes ocasionales sólo pueden tener 2. Los datos que componen la ficha de un usuario son su DNI y su nombre.

La biblioteca presta dos tipos de documentos: libros y revistas. La ficha de un documento se compone un código alfanumérico (que permite identificar el documento) y de su título. Además, la ficha de un libro tiene otro campo más: su año de publicación.

La duración máxima del préstamo de los libros a los socios es de 30 días, mientras que para los usuarios ocasionales la duración máxima del préstamo de los libros es de 15 días. La duración máxima del préstamo de una revista a un usuario es un tercio de la duración máxima del préstamo de un libro a ese mismo tipo de usuario.

Casos de uso:

**Selecciona documento**

**Presta documento actual**

**Devuelve documento actual**

**Busca documentos**

**Informe de préstamos**

**bibliotecario**



Una aplicación real incluiría también casosde uso para añadir y eliminar documentos, añadir y eliminar usuarios, etc. No se incluyen en el problema para limitar su complejidad)

A continuación, se procede a describir los casos de uso. No se entra en detalles de la interacción entre el bibliotecario y la aplicación (punto 1 de cada caso de uso), puesto que no va a ser tarea del alumno desarrollar esa parte.

***Caso de uso "Selecciona documento":***

1. El bibliotecario elige la opción "selecciona documento" e introduce el código del documento.
2. La aplicación pone como documento actual el documento con ese código.

- En el caso de que no exista ningún documento con ese código se notifica ***Caso de uso "Presta documento actual":***

1. El bibliotecario elige la opción "presta documento actual" e introduce el DNI del

usuario al que se va a prestar el documento actual.

1. La aplicación comprueba que es posible prestar el documento actual al usuario (existe un usuario con el DNI indicado, el usuario no ha alcanzado el límite de préstamos y el documento actual no se encuentra prestado)

- si no es posible prestar el documento, lo notifica y finaliza el caso de uso

1. La aplicación registra el documento como prestado al usuario ***Caso de uso "Devuelve documento actual":***
   1. El bibliotecario elige la opción "devuelve documento actual"
   2. La aplicación finaliza el préstamo del documento actual

- en el caso de que el documento actual no se encontrase prestado, se notifica y finaliza el caso de uso

***Caso de uso "Busca documentos":***

1. El bibliotecario elige la opción "busca documentos" e introduce un texto a buscar.
2. La aplicación busca todos los documentos que contengan ese texto en su título
3. La aplicación muestra los documentos encontrados ***Caso de uso "Informe de préstamos":***
   1. El bibliotecario elige la opción "informe de préstamos".
   2. La aplicación muestra por consola la información relativa a los documentos prestados: título, año de publicación (si es un libro), código, plazo de préstamo y usuario al que se le ha prestado. El formato debe ser exactamente el mostrado a continuación (respetando la ordenación por columnas).

El lenguaje de programación Java prestado a:12345678A (Socio)

El nombre de la rosa

(2OOO) Cód:P1O1

Plazo:3Odías

(198O)

Cód: A22

Plazo:15días

prestado a:98765432C (Usuario Ocasional)

Science

prestado a:11223344G (Usuario Ocasional)

...

Cód:

D1

Plazo: 3días

***Codigo en el archivo adjunto.***

El alumno deberá desarrollar las clases que proporcionan las operaciones que permitirían al programa principal implementar los casos de uso descritos con anterioridad (dichas clases deben gestionar las listas de usuarios y libros, la situación de prestado o libre de los documentos, la información sobre los documentos prestados a un usuario, etc.).

Se pide:

* Diseño arquitectónico de la parte de la aplicación encargada al alumno
* Código de las clases correspondientes a la parte de la aplicación encargada al alumno

(Utilizar la herencia en la implementación de las clases correspondientes a los documentos y los usuarios).

}